**CISCO VPN in Linux:**

sudo apt-get install network-manager-vpnc

sudo vi /etc/resolv.conf

nameserver 127.0.1.1

**Application Under Test:**

<https://uk.qa.thomascook.io/>

UK(Br Eng)

BE(Deutche/French)

NL(French)

Airtours

Packages - accommodation page

QA (Unit Testing) Our Domain

Integration

Staging

Production

application :

Angular Node JS( Frameworks)

features - regression/ uat

Given: Take a string search in the project directory

**Github project**

Go to github

<https://github.com/ThomasCookOnline/DigitalAutomationTesting>

Search Branch (use drop down) -> Compare

git log

git diff file\_name to see the changes made in the file

Squashing in case of many commits ( Doc will be provided on access)

**Git Repository:**

<https://github.com/ThomasCookOnline/DigitalAutomationTesting>

git@github.com:ThomasCookOnline/DigitalAutomationTesting.git

git@github.com:performancematters/regressionAutomation.git

**For SSH error:**

sign\_and\_send\_pubkey: signing failed: agent refused operation

**Execute below command:**

ssh-add

**Git Cloning:**

Change the directory path to the required directory path:

cd \d D:\..\..\

Copy the repo path from GIt site:

git clone<https://github.com/ThomasCookOnline/DigitalAutomationTesting.git>

git@github.com:ThomasCookOnline/DigitalAutomationTesting.git

git@github.com:ThomasCookOnline/components

Open the cloned project in sublime text editor/any other editor.

**Change the ownership & group:**

sudo chown maheshb \*

sudo chgrp maheshb \*

**Install the chromedriver & selenium server locally:**

~/ThomasCook/DigitalAutomationTesting/node\_modules/protractor/node\_modules/webdriver-manager/bin

node webdriver-manager clean

node webdriver-manager update

**To avoid git asking credentials on every pull/push:**

vi .git/config

Change following https Url in ssh format:

url = https://github.com/ThomasCookOnline/DigitalAutomationTesting.git

url = git@github.com:ThomasCookOnline/DigitalAutomationTesting.git

**OR**

1. **Configuration of git:**

**// File where configuration settings are saved.**

vi .git/config

Add user name password.

git config --global user.name "Mahesh Bhosale"

git config --global user.email "Mahesh\_Bhosale@epam.com"

Check you global variables

git config --global --list

//Example

url = [https://**mahesh-bhosale87:shiva789**@github.com/ThomasCookOnline/DigitalAutomationTesting.git](about:blank)

**//To add editor in Windows gitbash to avoid error while doing commit --amend:**

git config –global core.editor notepad

1. **To add the origin on remote:**

git remote add origin [git@github.com:mahesh-bhosale87/PhpTravels.git](mailto:git@github.com:mahesh-bhosale87/PhpTravels.git)

for following error that suggests that the remote named ‘origin’ already exist in local repository & can’t be added again.

fatal: remote origin already exists.

git remote set-url origin git@github.com:ppreyer/first\_app.git

git remote -v

git push origin master

**On Master:**

Git pull origin master

Git fetch --all

//Gruntfile - Configuration Task file

git status

git add --all .

git commit -m "Message"

git push origin branch\_name

**The more preferable push syntax for the first time push to set up upstream remote branch:**

git push -u origin my\_branch

**To unstage a file:**

git reset --mixed HEAD file\_name\_staged

**OR**

git reset HEAD file\_name\_staged

**OR**

git reset file\_name\_staged

**OR**

git rm --cached file\_name\_staged

**Alternate explicit syntax:**

git branch --set-upstream-to origin/my\_branch

**Everyday Updating of Branching:**

git checkout master

git pull origin master

git checkout

git pull --rebase origin master

**To rebase on pull to have your work pulled on the top of the upstream branch (like master):**

git pull --rebase

**OR**

If whitespace issues are commited

git pull --rebase --whitespace=fix

In case of conflict in the rebase resolve it & continue:

git rebase --continue

OR update .git/config

git config branch.master.rebase true

**git config --global branch.autosetuprebase always**

[branch]

autosetupmerge = always

autosetuprebase = always

l autosetupmerge controls whether git branch and git checkout -b imply the --trackoption, i.e. with your setting of always, git checkout branchname, if branchname exists on a remote but not locally, will create branchname tracking its remote counterpartgit checkout -b newbranch will create a new branch newbranch tracking whichever branch you had checked out before issuing this command

l autosetuprebase controls whether new branches should be set up to be rebased upon git pull, i.e. your setting of always will result in branches being set up such that git pullalways performs a rebase, not a merge. (Be aware that existing branches retain their configuration when you change this option.)So it makes perfect sense to have both autosetupmerge = always and autosetuprebase = always; in fact, that's also what I have.

**Create new branch:**

git checkout -b **TESTAUTO-409**\_Cancell...

git checkout -b **new\_branch\_name from\_branch\_name**

git branch branch\_name

**To see the difference between latest file pulled using pull:**

git diff master@{1} master file\_name

**To see the only difference of probable white space errors in the file not yet staged with staging area:**

git diff --check file\_modified\_not\_staged

**To see the changes introduced by the branch from its common ancestor from the other branch:**

git diff branch\_name...branch\_name

**OR**

git diff branch\_name ^branch\_name

**To see the difference between the local branch & its fetched remote upstream branch in case local branch is having updated commits:**

git diff branch\_name ^remote\_name/branch\_name

**Note: 3 dots**

**To see the difference between the local branch & its fetched remote upstream branch:**

git log branch\_name..remote\_name/branch\_name

**OR**

git log branch\_name --not remote\_name/branch\_name

**To see what you’re about to push to a remote:**

git log origin/master..HEAD

**OR**

git log origin/master..

**To see the details of changes introduced in the commits in log:**

git log -p

git log branch\_name --not remote\_name/branch\_name -p

**To bundle the commits in the form of file/directory from a branch that acts as repo:**

git bundle create mybundle.bundle HEAD master

**To bundle the selected commits in the form of file/directory from a branch that acts as repo:**

git bundle mybundle.bundle master ^SHA1\_of\_the\_commit\_next\_to\_which\_to\_be\_considered

**To unpack the bundle in the form of repository:**

git clone mybundle.bundle repo

**Verify before the cloning the bundle:**

git bundle verify ../mybundle.bundle

**To list the bundle branches:**

git bundle list-heads ../mybundle.bundle

**Fetch/pull commits from the bundle branch to local branch:**

git bundle fetch ../mybundle.bundle bundle\_branch:local\_branch

**Shorthand to add patch to the staged file:**

git add --patch

**Stage everything:**

git add --all .

**Interactive add to staging in case of number of patches are more & complicated:**

git add -i

**Change/Rename the branch name on local & remote:**

git branch -m old-name new-name

**To see the commits that can be pushed after the last push on the current branch:**

git show origin/master..HEAD

git show HEAD --not origin/master

**To see the default pull/push connection between remote & local branches:**

git remote show remote\_short\_name

git remote show origin

**AND**

git push origin :old-name new-name

git push -u origin new-name

**OR**

git push -u origin :old-name new-name

**testSupport/navigation/cucumber/commonNavigation.js**

**To set the branch to track the changes on remote/origin/branch:**

git checkout T8581899 //Local branch

git branch --set-upstream-to origin/T8581899

**To Reset log history:**

git reflog

git reset --hard HEAD@{5}

**Configure Git to set upstream branches automatically**

git config --global branch.autosetupmerge always

**To see all branch sync-up with their upstream**

git branch -vv

**In case the branch is behind the intended remote upstream branch rebase it by:**

git rebase

**(by default git uses upstream branch)**

If you want to rebase with other branch for e.g. origin/master:

git rebase origin/master

**To undo the modifications in the file:**

git checkout -- file\_name

**To see the log in graphical way:**

git log --oneline --decorate --graph -6

**To stash patches selectively based on individual files:**

git stash save -p “recalling\_name”

**To squash all related commits into one commit:**

git log

git rebase -i HEAD~4

**Alternate smarter squashing:**

git log

git reset --soft HEAD~4

git commit --amend

**Reset in detail:**

git reset file\_name

**=**

git reset --mixed HEAD file\_name

HEAD refers/points to the current commit in the log history. It effectively moves the staged file to tracking & loads HEAD commit state to staging.

**To copy file/checkout file from one branch to other:**

git checkout new\_branch

git checkout old\_branch file\_named

git commit -m 'Add file foo to gh-pages.'// Maybe Optional

**Wrong Commit Reset:**

git reset --hard HEAD^

git push -f

**For github change branch look into the following type of message:**

[elamperti](https://github.com/elamperti) wants to merge 2 commits into master from WEB-34292

**\* UI based comparison**

gitk

To add all files to staging area:

git add .

To undo git add . use git reset (no dot)

To undo git add on the files added to staging:

git reset file\_name

To setup the remote branch to track the changes on the local branch:

git branch --set-upstream my\_branch origin/my\_branch

**Git Merge Conflicts:**

**To abort merge:**

git merge --abort

**To revert branch to its pre-merge state in case merge conflict is resolved & merge commit is created:**

git reset --hard HEAD~

**Git merge conflict markers with base marker along with ours & their markers:**

git config --global merge.conflictstyle diff3

**To see the conflicts on the files involved in the merge:**

git log --oneline --left-right --merge

**OR run following after merge conflict occurred**

git diff

**To ignore white space differences during merge:**

git merge -Xignore-space-change branch\_name

**Mergetool:**

git mergetool

**After fixing the merge conflict commit to create merge commit:**

git add file\_name

**OR**

git add -all

git commit

**After fixing the rebase conflict:**

git add file\_name

**OR**

git add -all

git rebase --continue

**To remember the merge/rebase conflicts resolutions**

git config --global rerere.enabled true

**To see the difference of file before staging AND before/after merge conflict resolution:**

git rerere diff

**To create the conflicted file state which rerere has resolved:**

git checkout --conflict=merge file\_name

**To get it resolved again:**

git rerere

**To bisect the Good & Bad commits that introduce bug:**

**git bisect start**

Mark the current commit as bad:

**git bisect bad**

Find the last known working commit (git log maybe) & mark it as an good commit:

**git bisect good commit\_sha1**

Keep on doing this until you find the one at the root & then reset t move HEAD to original position:

**git bisect reset**

**In most cases update the local branch with the remote branch(at left) & save. Then commit.**

git commit

git pull

**Application Flow:**

Home/Search page: Destination search

Search Resul-> Details -> Accomodation

Package details ( Flight & room details)

-> Get Quote after making the selection

-> Continue -> PAX Page

-> Payment Page -> Pay in Full/in installment (Right pane displays all selection details) -> Accept & confirm

**Node install:**

<https://nodejs.org/en/download/package-manager/>

**Install Node.js on Linux:**

sudo apt-get install nodejs

(Note: installing it as sudo may hardly help you try not to install it with sudo)

**Install Protractor Node with admin rights (sudo su):**

npm install -g protractor

npm install -g webdriver-manager

**Uninstallation**:

Uninstall the corrupt packages:

npm remove -g grunt-cli

Uninstall the node:

sudo apt-get remove nodejs

**Get the chrome driver for protractor node:**

sudo webdriver-manager start

sudo webdriver-manager update

**Install grunt:**

npm install -g grunt-cli

**To install all the project dependencies:**

npm install

**VPNC commands to get connected to the VPN:**

vpnc

vpnc-disconnect

Edit /etc/vpnc/default.conf

**Project Alignment:**

**Directories**:

features: ( contains feature files for various modules like - regression)

Feature file naming convention “ticket\_number.feature” e.g. c10192.feature

Search the part of the step description in step\_definitions directory to find matching step definition (which happens to be an js module). Step definition module contains function object with step identifier (Given/When/Then) as their function properties.

**Observations of feature files:**

Most of the files contain only one scenario as against the expectation of 5 to 25 scenarios per feature file.

Scenario names may not be changed/updated while copying them to new feature files.

Feature description needs to be added in this manner:

**In order to** buy beer

**As** an account holder

**I want to** withdraw cash from the ATM

Comment out steps(#) in the debugging.

Scenario name should be concise & expressive also durable over the changes in the behavior of the system.

**Tip**: Use Context(Given) and event(When) to summarize into scenario name. Avoid using reference to outcome (Then) for scenario name.

Description - Can be placed after each keyword - ideal for documentation for stakeholders (concrete example description)

Comment (#) - should be used to add some technical details for testers/programmer. Useful in debugging.

Step definition methods match step text irrespective of the step keywords (Given/When/Then) so make difference between the step texts for different type of step keyword.

Do not use .\* in capture group since \* is greedy operator it can accept any value that may not be valid in the given set of acceptable values. Most useful group capture for string is (\w+). For numeric it should be (\d+)

Use ? To make previous character optional

**Regular Expression Tut:**

<http://regexr.com/>

: It is useful in case of multiple scenarios with common steps (precondition actions). Current framework doesn’t create multiple scenarios in the same file?

Note: Tags must not be applied to the **background**.

There are two uses of data tables:

1 In abstraction of multiple similar substeps(AND or BUT) with different data can be summarized in one step + data.

2. In Examples of Scenario Outline to convert it into data driven scenario mechanism.

Examples should be divided as per the rules applicable to the type of input data. Rule can be the description of the given example. Example should have a short name which should be abbreviation of the underlying rule.

Grouping of steps in step definition should be avoided instead write a high level step & create a separate step definition using the framework method.

**Search**

**Packages**

**Command to run Test Case locally:**

**Two different syntaxes:**

grunt e2e-cuke --env=qa --market=uk --suite=regressionUK --tags=@c15072

grunt e2e-cuke --env staging --market be --suite uatBE --tags @c397666

Market & env are used to configure browser.

Suite & tags are used to filter out the feature file.

Valid suite name can be found in the following file:

testSupport -> config -> protractor

Be market has two languages:

1.Dutch 2.French

For UAT folder environment: staging

For other folders environment: qa

**Sublime Text Editor:**

Shortcut key

for console: Ctrl+`

For command pallette: Ctrl + Shift + P

Select entire line: Ctrl +L

Delete entire line: Ctrl + X

Move to the definition of field: F12

Revert to previous location: Alt + -

Preferences -> Package Control

It launches a package control pop-up with its command list. Use appropriate command to add the plugins.

E.g. Install Package SyncedSideBar (You need to click on the particular command & set the argument from the consequent list)

# SublimeCodeIntel //for code intelligence

[ClickableURLs](https://github.com/leonid-shevtsov/ClickableUrls_SublimeText2) //for clickable urls

[SideBarEnhancements](https://github.com/titoBouzout/SideBarEnhancements/tree/st3) //More sidebar options in context menu(right click)

Git //Git integration

ayu // theme<https://github.com/dempfi/ayu>

AlignTab //Align lines

<https://www.granneman.com/webdev/editors/sublime-text/packages/how-to-install-and-use-sublime-alignment/>

**Project Process:**

\* PR stands for pull request.

UAT testing has staging environment.

Assigned mail

JIRA

TestRail link

Credentials

Create branch as per JIRA ticket number

Review Phase

**Jenkins Job Configuration**

Links of interest:

<https://jenkins.thomascook.io/>

<https://jenkins.thomascook.io/view/e2e%20regression%20jobs/>

Create Job:

<https://jenkins.thomascook.io/view/e2e%20regression%20jobs/job/web-ui-e2e-regression-tests-win-chrome/>

Click Build With parameters.

GIT\_COMMIT: add your branch name

TAGS: enter you test tag

SPECS: dropdown select test/cucumber/features/smoke/\*.feature

SUITE: smoke

MARKET: nl

ENV: same as integration

click on build

click on the link for todays date in build history and link on cucumber-jvm reports

thr url you get will be your jenkins link for the PR

Create Pull Request

Comment should contain:

JIRA:

TestRail link:

Description:

Jenkins Cucumber JVM Report link:

<https://jenkins.thomascook.io/view/e2e%20regression%20jobs/job/web-ui-e2e-UAT-stage-win-chrome/cucumber-html-reports/>

**Page Objects:**

**Steps are written with 2 Table field conventions:**

1. With **>**

2. with **space**

**between** words

> translates to . (Object with chained fields)

icon > forType > pencil == icon.forType.pencil

Step definitions in the following folders refer to the page objects.

Step\_definitions ( -> languageSteps for e.g.)

testSupport -> navigation -> urlNavigation.js

global.isMobile?

PageObjects-> pageComponents -> component.js

It is a common wrapper for all the locators.

It accepts locator value in its constructor function.

PageObjects-> pageComponents -> formField.js

It contains **config** object which is representative of the locator object that contains properties like field(CSS locator of the actual element field), label, error etc.

**Feature file with field validation on particular page -> genericExpectationSteps.js**

1. genericExpectationSteps.js

expect.tableToMeetExpectations({ **page**: page, **field**: field, table: table});

2.multiExpectations.js

tableToMeetExpectations: **function** (config) {

:

**var** verifyFunction = *\_getVerifyFunction*(config);

3.multiExpectations.js

verifyFunction = *\_rowToMeetExpectation*;

4.multiExpectations.js

**return singleExpects**.*stepToMeetExpectation*({

5.singleExpectations.js

pageComponent = getPageComponent(config);

6.utils.js

pageComponent = **lookup**.pageComponentByName(config.page, config.**field**, config.context);

7.lookup.js

pageComponentByName: **function** (parentObj, lookupName, context) {

:

**var** fieldData = *\_getFieldData*(lookupName, lookupName, context);

8.lookup.js

**function** *\_getFieldData*(fieldName, lookupName, context) {

:

**var** fieldData = {**componentType**: **'field'**, **fieldName**: fieldName, **lookupName**: lookupName, **context**: context};

Structure of the fieldData

{ componentType: 'field',

fieldName: 'Price Ticket > Price Ticket > Bottom',

lookupName: 'Price Ticket > Price Ticket > Bottom',

context: 'table row 1',

objectName: 'priceTicket.priceTicket.bottom',

parentPath: [ 'priceTicket', 'priceTicket' ],

name: 'bottom' }

fieldData.**objectName** = **conversions**.toObjectFieldName(fieldData.fieldName);

fieldData.**parentPath** = fieldData.**objectName**.split(**'.'**);

fieldData.**name** = fieldData.**parentPath**.pop();

8.1 conversions.js

toObjectFieldName: **function**(string) {

**return stringUtils**.replaceFeatureDelimiters(string, **'.'**).*split*(**'.'**).map(\_.*camelCase*).join(**'.'**);

},

**This maps expectationFunction to:**

'equals': **function** () {

**return** expect.toEqual(pageComponent, value, fieldData.componentType, fieldData.fieldName, index);

}

**var** utils = require('./utils');

**var** getPageComponent = utils.getPageComponent;

**Utils.js:**

**function** getPageComponent(config) {

…

pageComponent = lookup.pageAndComponentByName(config.page + ' ' + config.field, config.context).component;

**Lookup.js**

pageAndComponentByName: **function** (friendlyName, context) {

…

**var** pageObject = exports.pageObjectByName(pageName);

pageObjectByName: **function** (friendlyName) {

…

**var** pageObject = pageObjects[pageObjectName];

**var** pageObjects = require('./pageObjects');

It is the expectation type in feature file sentence (e.g. equals, contains etc.)

testSupport -> expectations -> expect.js/multiExpectations.js

It takes the data table of expectations in the Then method &

PageObjects -> pageComponents.js

**(?) lodash.isObject(arg)**

**debug** exposes a function; simply pass this function the name of your module, and it will return a decorated version of console.error for you to pass debug statements to. This will allow you to toggle the debug output for different parts of your module as well as the module as a whole.

var debug **=** require('debug')('http')

, http **=** require('http')

, name **=** 'My App';

*// fake app*

debug('booting %s', name);

Array map() method:

The map() method calls the provided function once for each element in an array, in order.

Lodash documentation:

<https://lodash.com/docs/4.17.4#endsWith>

**Eclipse Plugin to run grunt tasks:**

<https://www.eclipse.org/community/eclipse_newsletter/2016/may/article4.php>

Eclipse Grunt / Gulp tools use the system installation, hence the following software must be pre-installed:

[Node.js](https://nodejs.org/en/)

[npm](https://www.npmjs.com/)

gulp-cli (npm install -g gulp-cli)

grunt-cli (npm install -g grunt-cli)

Webeclipse plugin for terminal+ & other.

Bebel for syntax of ES6

SummitEditor prereq for

[**npm**](https://www.npmjs.com/)is a package manager for **Node.js** with hundreds of thousands of packages. Although it does create some of your directory structure/organization, this is not the main purpose.

The main goal, as you touched upon, is automated dependency and package management. This means that you can specify all of your project's dependencies inside your **package.json** file, then any time you (or anyone else) needs to get started with your project they can just run **npm install** and immediately **have all of the dependencies installed**. On top of this, it is also possible to specify what versions your project depends upon to prevent updates from breaking your project.

It is definitely possible to manually download your libraries, copy them into the correct directories, and use them that way. However, as your project (and list of dependencies) grows, this will quickly become time-consuming and messy. It also makes collaborating and sharing your project that much more difficult.

Hopefully this makes it more clear what the purpose of npm is. As a Javascript developer (both client-side and server-side), npm is an indispensable tool in my workflow.

Eclipse Download:

<https://www.eclipse.org/downloads/eclipse-packages/>

Download following version for JS (Node.js support):

[Eclipse IDE for JavaScript and Web Developers](http://www.eclipse.org/downloads/packages/eclipse-ide-javascript-and-web-developers/neon2)

Watch out videos from for eclipse feature learning:

<https://www.eclipse.org/neon/>

To install all the dependencies in the eclipse project:

package.json -> right click -> run as -> npm install -> node\_modules

Steps (cucumber): features folder

This is executed with the help of grunt task

Step Definitions(Node.js ->Cucumber.js)

1. WebServices **to reach** directly on the specific page step\_definitions/givenSteps.js

2. Other step definitions specific to the **actions** on the page.

Page Objects(Protractor.js atop jasmine framework):

Component is parent Object of formField object.

formField.js contains FormField object which is set to have parent object Component.

testSupport folder

**Webstorm:**

To resolve grunt port conflict in debugging:

Add below at the start of code (after ‘use strict’)

process.execArgv = [];

Note: to make debugging working, add “process.execArgv = [];” to the top of Gruntfile.js and change node\_modules\grunt-express-server\tasks\lib\server.js to options.opts.unshift('–debug-brk');

Key:

<http://us.idea.lanyus.com/>

If you get a "The license CNEKJPQZEX has been cancelled.", you can append "0.0.0.0 [account.jetbrains.com](http://disq.us/url?url=http://account.jetbrains.com:YO6WUGeUP2FpHMY51uWaNe68Y8w&cuid=4737681)" to hosts file. /etc/hosts

**Jenkins local:**

**Jenkins:**

Installation:

1. Download & install Git Bash in Windows to run unix specific commands.

2. Download latest jenkins.war

3. Launch git bash on the downloaded path & run command:

java -jar jenkins.war

java -jar Jenkins/jenkins.war --httpPort=8081

4. .jenkins is the jenkins home directory (**Refer Page 91**)

5. Jenkins will be installed & need a password for authentication. Password will be at location similar to:

C:\Users\mbhosale\.jenkins\secrets\initialAdminPassword **OR**

cat ~/.jenkins/secrets/initialAdminPassword

<https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins#InstallingJenkins-WindowsInstallation>

Since Jenkins was written to work on unix-like platforms, some parts assume the presence of unix-utilities. It is advised to install these as well on Windows.

Install Standalone Jenkins package for windows.

Start link for above jar command: [http://localhost:8081/](http://localhost:8080/)

Latest Jenkins Version 2.64

**Important options on the Jenkins Dashboard:**

**Restart is enabled on the Jenkins installed as windows service.**

**1. Manage Jenkins -> Configure System:**

It covers below topics:

Maven

Docker

Jenkins location

Configuration of Version Control System (Git/Subversion) Plugin

Git Plugin:Global config.email config.name for git plugin

CVS 3 compression levels:

1. Private Key Location

2. Private Key Password

3. Known Hosts Location

Email Configuration

Email Notification: SMTP Server = smtp.gmail.com

Extended Email Notification: Default Recipient = bhosalemahesh87@gmail.com

**2. Manage Jenkins -> Global Tool Configuration:**

Plugins & JDK installations required for Builds:

**JDK**

**Git**

**Gradle**

**Maven (Build Script Framework)**

**Ant (Build Script Language)**

**NodeJS** You can install specific nodejs versions from here & assign the name to that installation which can be referred in Build Job for its configurations.

**Docker etc.**

Number of options depends on the plugins downloaded from **Manage Plugin** options.

If you download new plugin from **Global Tool Configuration** or **Manage Plugin** for the first run it will be stored in tools directory in .jenkins directory (jenkins home directory).

**Java:**

For this need to have Oracle Credentials to install Java.

**for the first run it will be stored in tools directory in .jenkins directory**

[**Role-based Authorization Strategy**](https://wiki.jenkins.io/display/JENKINS/Role+Strategy+Plugin)

It is used to allocate access rights to different groups of people depending on their authority in the project.

**Powershell:**

For build actions if jenkins is setup on windows.

**html publisher:**

To publish html reports

[**Green Balls**](https://wiki.jenkins.io/display/JENKINS/Green+Balls)

To show build status as green when successful

**Maven:**

Use urls similar to following:

http://redrockdigimark.com/apachemirror/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.zip

**3. Manage Plugins** **:**

Jenkins -> Manage Jenkins -> Manage Plugins

(This is for setting the infrastructure for the project)

Application/project building requires build tools that will manage dependencies for the project. Maven is one such build tool.

Jenkins build tool can configure application build tools like maven.

**Proxy Setting:**

On Advanced Tab:

Server: 10.220.2.251

Port: 3128

Username: admin

Password: [secrets/initialadminpassword]

**4. To create New Build job:**

Jenkins->New Item

For simple jobs select ‘Freestyle Project’

Give the name without space.

To change the workspace -> General Tab -> Advanced -> Use custom workspace

**-> Source Code Management:**

Source Code may be located remotely at centralized location for version control. It may be referred by the web link. This needs to be managed.

Note: This is the binding point between the source code & jenkins. If the source code is maintained under VCS like git. Jenkins ‘build triggers’ can be configured to trigger the build on each code push (change).

**Git: (Page 124)**

Specify repository URL. It will clone the repository in the directory .jenkins/jobs/job\_name/workspace

Branches to build if left blank or set to \*\* will build all the remote branches

Additional Behaviours

Clean after checkout

Prune stale remote tracking branches

Merge before build

Use commit author in changelog

**-> Build Triggers:**

Decides about when to trigger the build. You can use unix ‘cron’ to define ‘Poll SCM’ intervals

When you need to trigger the jobs manually simply keep this section empty.

**-> Build Environments:**

Abort the build if it's stuck

Add timestamps to the Console Output

Provide Node & npm bin/ folder to PATH

**-> Build:**

Decides about which build tool to be used. Subsequently it allows the build steps (goal/task) to be declared for the same build tool.

Execute Shell script

**-> Post Build Actions:**

Decides about outcome of the Build. It includes reporting results of the build/notifications of the build/Archiving the latest application artifacts.

Cucumber reports

E-mail notification

Discard old builds plugin

Git Publisher (important)

**Parameterized Build:**

It requires parameterized build plugin.

Use choice parameters where selection option is to be provided.

<http://localhost:8080/configureTools/>

Set the path to the locally installed Git (for faster execution)

\*

**To get unbroken HTML reports execute following in Manage Jenkins -> Script Console:**

|  |
| --- |
| System.clearProperty("hudson.model.DirectoryBrowserSupport.CSP");  System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", "sandbox allow-same-origin allow-scripts; default-src 'self'; script-src \* 'unsafe-eval'; img-src \*; style-src \* 'unsafe-inline'; font-src \*"); |

**Aauthorities:**

**Plugin**

role-based authorization

Jenkins is an open source DevOps tool that will help you to deploy and automate your enterprise application.

In Jenkins, by default you can create users, but not groups.

So, if you want groups in Jenkins, you have the following few options:

§ Use OpenLDAP with Jenkins

§ Use Active Directory with Jenkins

§ Use Unix user/group database. This will use PAM library to integrate with Jenkins.

§ Use “Role-based authorization strategy” plugin for Jenkins

The default behavior (i.e Can’t create group) is because it uses Jenkins user database for the security realm.

To verify this, login to Jenkins as admin, go to “Manage Jenkins”, click on “Configure Global Security”, and under the “Access Control” section, for the “Security Realm”, if you’ve selected “Jenkins’ own user database”, then you can only create users, and not groups.

If you want the ability to create groups (roles) and assign certain privileges to certain roles, and if you are not sure which option to choose, then probably using Jenkins role-based plugin is the best choice.

The role-based authorization strategy plugin works very well with the default Jenkins own user database, and the matrix authorization strategy for projects.

If you are using project based matrix authorization strategy, you can restrict user and group access on a job by job basis. i.e Each and every project will have its own restrictions.

We discussed about this in detail earlier: [How to Setup User Security on Jenkins with Project Matrix Authorization](https://www.thegeekstuff.com/2016/06/jenkins-security/)

In this tutorial, we’ll focus on how to setup role based restrictions for Jenkins Jobs.

## Install Role-based Authorization Strategy Plugin

Login to Jenkins with your admin account -> Click on “Manage Jenkins” -> Click on “Manage Plugins” -> Click on “Available” tab -> Search for “role” in the Filter text box.

You’ll see “Role-based Authorization Strategy” in the results. Click on the “check-box” in front of it to select this item. Click on “Install without restart” button at the bottom.

Restart the Jenkins after this plugin is installed.

Once the plugin is installed, you should see it under “Installed” tab as shown below.

## Change Jenkins Authorization Method

Once the plugin is installed, next step is to change the default Jenkins authorization method to use this role-based plugin.

For this, go to “Manage Jenkins”, click on “Configure Global Security”, under the “Access Control” section, for the “Authorization”, click on “Role-Based Strategy” as shown below.

## Manage and Assign Roles Options

Now, if you go to “Manage Jenkins”, you’ll see “Manage and Assign Roles” as shown below.

When you click on “Manage and Assign Roles”, you’ll have the following three choices:

## Create a New Global Role

Click on “Manage Roles”. From here you can create global roles that will be applicable for all the objects in the jenkins. For example, you can create “admin” roles, “developer” roles, “devops” role, etc here.

To add a global role, enter the name (for example: developer), and click on “Add” as shown below.

Once you add a global role (for example: developer), then select the permissions that you want to assign for that particular global role.

The following are the permissions available to be assigned to your new global role.

§ Overall – Administer, ConfigureUpdateCenter, Read, RunScripts, UploadPlugins

§ Credentials – Create, Delete, ManageDomains, Update, View

§ Agent – Build, Configure, Connect, Create, Delete, Disconnect, Provision

§ Job – Build, Cancel, Configure, Create, Delete, Discover, Move, Read, Workspace

§ Run – Delete, Replay, Update

§ View – Configure, Create, Delete, Read

§ SCM – Tag

Note: Don’t forget to click on the “Save” button at the bottom of this page to save your changes.

## Create a New Project Role

Here, you can create roles that can be applied to only certain projects (i.e jobs) that match a certain pattern. For example, you can create a project role called “web”, which will apply only to all the projects (i.e jenkins jobs) that starts with the keyword “web\*” (this is the pattern) as shown below.

The following are few things to keep in mind about the project role pattern field:

§ You’ll enter a regular expression in this field. For example, web\* will match all the Jenkins jobs that starts with “web”.

§ By default, this value is case sensitive

§ If you want case-insensitive, add “(?i)” to the pattern. For example (?i)web\* will match jobs starting with both “web” and “Web”.

Once you’ve added a project role (for example: db, or web, or php-project), then select the permissions that you want to assign for that particular project role.

The following are the permissions available to be assigned to your new project role.

§ Credentials – Create, Delete, ManageDomains, Update, View

§ Job – Build, Cancel, Configure, Create, Delete, Discover, Move, Read, Workspace

§ Run – Delete, Replay, Update

§ SCM – Tag

Note: Don’t forget to click on the “Save” button at the bottom of this page to save your changes.

Also, in the Slave Roles section, you’ll create slave roles, which will give the ability for you to assign node-related permissions. In most situations, you won’t be using this.

## Assign Users to the new Group (Role)

After creating the roles (either global role, or project role) with appropriate permissions, the next step is to assign users to this role.

For this, go to “Manage Jenkins” -> “Manage and Assign Roles” -> Click on “Assign Roles”.

Under the “Project Roles” section, you’ll see all the project roles that we created from the previous section. These roles will be listed as columns.

Now, to assign an user to any one of these project roles, first enter the name of the user in the “User/group to add” textbox (for example: lisa), and click on “add” button. Please note that this user should already be created in the Jenkins.

This will add the user as a “row” in this section. Now, click on the “check-box” corresponding to the project-role name for user “lisa”, which will assign that particular role to user “lisa”.

In the following example, I’ve added “java-project” and “php-project” Project roles to user lisa.

Cron Jobs:

By setting the schedule period to 15 13 \* \* \* you tell Jenkins to schedule the build every day of every month of every year at the 15th minute of the 13th hour of the day.

Jenkins used a [cron expression](https://en.wikipedia.org/wiki/Cron" \l "CRON_expression), and the different fields are:

1. MINUTES Minutes in one hour (0-59)
2. HOURS Hours in one day (0-23)
3. DAYMONTH Day in a month (1-31)
4. MONTH Month in a year (1-12)
5. DAYWEEK Day of the week (0-7) where 0 and 7 are sunday

If you want to schedule your build every 5 minutes, this will do the job : \*/5 \* \* \* \*

If you want to schedule your build every day at 8h00, this will do the job : 0 8 \* \* \*

For the past few versions (2014), Jenkins have a new parameter, H (extract from the [Jenkins code documentation](https://github.com/jenkinsci/jenkins/blob/master/core/src/main/resources/hudson/triggers/TimerTrigger/help-spec.jelly)):

To allow periodically scheduled tasks to produce even load on the system, the symbol H (for “hash”) should be used wherever possible.

For example, using 0 0 \* \* \* for a dozen daily jobs will cause a large spike at midnight. In contrast, using H H \* \* \* would still execute each job once a day, but not all at the same time, better using limited resources.

Note also that:

The H symbol can be thought of as a random value over a range, but it actually is a hash of the job name, not a random function, so that the value remains stable for any given project.

[More example of using 'H'](https://stackoverflow.com/a/21939671)

**JSON.stringify**

Used to convert object to string following JSON notation. functions that are properties of the object needs to be converted to string explicitly if you want it to be stringified otherwise it will be omitted.

This function is used to log the object details.

**Creating the field locators with CSS:**

Reference:<http://www.w3schools.com/cssref/css_selectors.asp>

E.g.

roomLead: components.FormField({field: '[id\*="Room' + room + '"][type="radio"]', label: '#room-occupant-' + pax + ' > div:first-child span'}),

|  |  |  |
| --- | --- | --- |
| [[attribute\*=value]](http://www.w3schools.com/cssref/sel_attr_contain.asp) | a[href\*="w3schools"] | Selects every <a> element whose href attribute value contains the substring "w3schools" |
| [element>element](http://www.w3schools.com/cssref/sel_element_gt.asp) | div > p | Selects all <p> elements where the parent is a <div> element  This selects only child elements in the hierarchy & omits all the next matching elements(child of child).  **Note: ‘space’ represents any element with matching criterion following to current element while ‘>’ represents parent & its direct children elements** |
| [:first-child](http://www.w3schools.com/cssref/sel_firstchild.asp) | p:first-child | Selects every <p> element that is the first child of its parent |

GIF recorder:

sudo apt install gifsicle luarocks

sudo apt install libgirepository1.0-dev

luarocks install --server=http://luarocks.org/dev gifine

Gifine

Jenkins:

CI Build Automation Tool.

It is a Java Web Application

Maven:

Java Build Tool

Prerequisites:

Java

Git

Github Account to have stable version of the project (that follows CI practices)

**To run the jenkins application navigate to the directory with jenkins application file & run:**

Java -jar jenkins.war

To avoid port conflicts:

java -jar jenkins.war --httpPort=8081 --ajp13Port=8010

java -jar ~/Jenkins/jenkins.war --httpPort=8081 --ajp13Port=-1 --controlPort=8001

For outlook mails to work well:

java -jar  -Dmail.smtp.starttls.enable=truejenkins.war --httpPort=8081 --ajp13Port=-1 --controlPort=8001

**To see the admin password to login to the Jenkins:**

cat /home/maheshb/.jenkins/secrets/initialAdminPassword

**Jenkins as service installation(not recommended as it has git ssh credential issues):**

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins

sudo /etc/init.d/jenkins start

sudo /etc/init.d/jenkins stop

cat /var/lib/jenkins/secrets/initialAdminPassword

**Browser:**

http://localhost:8080/

JENKINS\_HOME = /var/lib/jenkins

**GitHub Servers:**

APIU URL:<https://api.github.com>

**Email Notification:**

<https://wiki.jenkins-ci.org/display/JENKINS/GMail>

Valid gmail smtp server: smtp.gmail.com

**Custom Workspace for the job:**

in the general tab, when you go to the "advanced options" , you will see the "custom workspace" option

Node Project:

Eclipse:

1. Install Nodeclipse plugin from eclipse:

<https://marketplace.eclipse.org/content/nodeclipse#group-details>

2. Create new project in eclipse:

New -> Node.js Express Project

**Protractor Main References:**

<https://github.com/angular/protractor/blob/master/docs/api-overview.md>

<http://www.protractortest.org/#/tutorial>

<http://www.protractortest.org/#/toc>

<https://semaphoreci.com/community/tutorials/getting-started-with-protractor-and-cucumber>

3. package.json:

<https://css-tricks.com/why-npm-scripts/>

{

"name": "MyPro",

"version": "0.0.1",

"private": **true**,

"**scripts**": {

"start": "node app.js"

},

"dependencies": {},

"devDependencies": {

**"protractor": "^4.0.0"**,

**"cucumber": "~1.2.1",**

"protractor-cucumber-framework": "^0.6.0",

"eslint-plugin-cucumber": "^0.1.0",

"express": "3.2.6",

"jade": "\*"

}

}

Here, protractor is installed as node in node\_modules repository of the project directory.

Protractor will in turn have webdriver-manager node in its bin directory which will cater to downloading selenium server jar & chromedriver by default.

The server acts as proxy between your test script (written with the WebDriver API) and the browser driver (controlled by the WebDriver protocols).

[Test Scripts] <-[Protractor]-<--- > [Selenium Server] < ------------ > [Browser Drivers]

**The --save and --save-dev install flags:**

The easier (and more awesome) way to add dependencies to your package.json is to do so from the command line, flagging the npm install command with either --save or --save-dev, depending on how you'd like to use that dependency.

**To add an entry to your package.json's dependencies:**

npm install <package\_name> --save

**To add an entry to your package.json's devDependencies:**

npm install <package\_name> --save-dev

**Package version syntax:**

X-Ranges 1.2.x 1.X 1.2.\* \*

Any of X, x, or \* may be used to "stand in" for one of the numeric values in the **[major, minor, patch]** tuple.

\* := >=0.0.0 (Any version satisfies)

1.x := >=1.0.0 <2.0.0 (Matching major version)

1.2.x := >=1.2.0 <1.3.0 (Matching major and minor versions)

Tilde Ranges ~1.2.3 ~1.2 ~1

Allows patch-level changes if a minor version is specified on the comparator. Allows minor-level changes if not.

Caret Ranges ^1.2.3 ^0.2.5 ^0.0.4

Allows changes that do not modify the left-most non-zero digit in the [major, minor, patch] tuple. In other words, this allows patch and minor updates for versions 1.0.0 and above, patch updates for versions 0.X >=0.1.0, and noupdates for versions 0.0.X.

Package Range Ref:

<https://docs.npmjs.com/misc/semver>

## Specifying Packages

To specify the packages your project depends on, you need to list the packages you'd like to use in your package.json file. There are 2 types of packages you can list:

"dependencies": these packages are required by your application in production

"devDependencies": these packages are only needed for development and testing

### Manually editing your package.json

You can manually edit your package.json. You'll need to create an attribute in the package object called dependencies that points to an object. This object will hold attributes named after the packages you'd like to use, that point to a [semver](https://docs.npmjs.com/getting-started/semantic-versioning) expression that specifies what versions of that project are compatible with your project.

If you have dependencies you only need to use during local development, you will follow the same instructions as above but in an attribute called devDependencies.

For example: The project below uses any version of the package my\_dep that matches major version 1 in production, and requires any version of the package my\_test\_framework that matches major version 3, but only for development:

{

"name": "my\_package",

"version": "1.0.0",

"dependencies": {

"my\_dep": "^1.0.0"

},

"devDependencies" : {

"my\_test\_framework": "^3.1.0"

}

}

**Config.js:**

Please note that if you set seleniumAddress, the settings for seleniumServerJar, seleniumPort, seleniumArgs, browserstackUser, browserstackKey, sauceUser and sauceKey will be ignored.

To run your tests against a remote Selenium Server, you will need an account with a service that hosts the server (and the browser drivers). Protractor has built in support for [BrowserStack](https://www.browserstack.com/) and [Sauce Labs](http://www.saucelabs.com/).

**Using Sauce Labs as remote Selenium Server**

In your config file, set these options:

sauceUser - The username for your Sauce Labs account.

sauceKey - The key for your Sauce Labs account.

Please note that if you set sauceUser and sauceKey, the settings for seleniumServerJar, seleniumPort, seleniumArgs, browserstackUser and browserstackKey will be ignored.

Protractor can test directly against Chrome and Firefox without using a Selenium Server. To use this, in your config file set directConnect: true.

directConnect: true - Your test script communicates directly Chrome Driver or Firefox Driver, bypassing any Selenium Server. If this is true, settings for seleniumAddress and seleniumServerJar will be ignored. If you attempt to use a browser other than Chrome or Firefox an error will be thrown.

The advantage of directly connecting to browser drivers is that your test scripts may start up and run faster.

Require.resolve(): It will resolve the path of the specified **single** file/module & return it. It will **not** return the module itself.

Reference:

<https://github.com/angular/protractor/blob/master/docs/server-setup.md>

<https://github.com/angular/protractor/blob/master/lib/config.ts>

<https://nodejs.org/api/globals.html#globals_require_resolve>

<https://github.com/angular/protractor/blob/master/docs/frameworks.md>

4. Linters:

Code [linting](http://en.wikipedia.org/wiki/Lint_(software)) is a type of static analysis that is frequently used to find problematic patterns or code that doesn’t adhere to certain style guidelines. There are code linters for most programming languages, and compilers sometimes incorporate linting into the compilation process.

JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed in order to find syntax or other errors. Linting tools like ESLint allow developers to discover problems with their JavaScript code without executing it.

The primary reason ESLint was created was to allow developers to create their own linting rules. ESLint is designed to have all rules completely pluggable. The default rules are written just like any plugin rules would be. They can all follow the same pattern, both for the rules themselves as well as tests. While ESLint will ship with some built-in rules to make it useful from the start, you’ll be able to dynamically load rules at any point in time.

ESLint is written using Node.js to provide a fast runtime environment and easy installation via [npm](http://npmjs.org/).

**"eslint-config-eslint"**: **"^3.0.0"**,

**"eslint-plugin-cucumber"**: **"^0.1.0"**,

5. To view hidden files like .eslintrc

From there, uncheck .\* resources.

So Package Explorer -> View Menu -> Filters -> uncheck .\* resources.

5. Cucumber eclipse plugin in “install new Software”:

Type name as you wish, let’s take “**Cucumber**” and type “**http://cucumber.github.com/cucumber-eclipse/update-site**” as location. Click **OK**.

6. Install Integrated TM Terminal in eclipse:

<https://marketplace.eclipse.org/content/tm-terminal#group-details>

7. Within the project (at ~/workspace/MyPro/node\_modules/protractor/bin):

node webdriver-manager update

node webdriver-manager start

8. Gherkin step definition generator:

<https://chrome.google.com/webstore/detail/tidy-gherkin/nobemmencanophcnicjhfhnjiimegjeo>

9. Synchronous/Asynchronous Behaviour of Protractor JS:

<https://spin.atomicobject.com/2014/12/17/asynchronous-testing-protractor-angular/>

Behind the scenes of WebDriver, each call that interacts with the browser, such as get(), findElement(), sendKeys(), and click(), is being scheduled and pushed onto the WebDriver ControlFlow, allowing us to not have to worry about using .then() on the resulting promises (unless we specifically want the result returned by the call).

10. Public/Private/Privileged/static members in JS function object:

<https://robertnyman.com/2008/10/14/javascript-how-to-get-private-privileged-public-and-static-members-properties-and-methods/>

// Constructor

function Kid (name) {

// Private

var idol = "Paris Hilton";

// Privileged

this.getIdol = function () {

return idol;

};

// Public

this.name = name;

}

// Public

Kid.prototype.getName = function () {

return this.name;

};

// Static property

Kid.town = "South Park";

11. function declaration vs function expression:

<https://kangax.github.io/nfe/#named-expr>

Expression: Can be in assignment(as function expression) /grouping”()”/new expression. Run time reference for grouping.

Declaration: plain declaration can be in other function. Parse time reference.

Statement : Assignment operation. It has expression. It can be considered for function variable hoisting. Parse time reference.

**First of all, function declarations are parsed and evaluated before any other expressions are**. Even if declaration is positioned last in a source, it will be evaluated foremost any other expressions contained in a scope.

Function Declarations are only allowed to appear in Program or Function Body. Syntactically, they can not appear in Block ({ ... }) — such as that of if, while or for statements.

Whenever function appears directly in a block (i.e. In {} of if while or for) it should actually be considered a syntax error, not function **declaration** or **expression**

10. Closures (Extensively used in NodeJS):

<http://javascriptissexy.com/understand-javascript-closures-with-ease/>

Closures get value access only when they are called (not when their definition is assigned to some var in another function)

To fix this side effect (bug) in closures, you can use an Immediately Invoked Function Expression (IIFE)

Closures have access to the outer function’s variable even after the outer function returns.Therefore, you can call the inner function later in your program

closures have access to the **updated** values of the outer function’s variables.

11. Callback functions:

<http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/>

Javascript Functions can be “stored in variables, passed as arguments to functions, created within functions, and returned from functions”.

We can pass functions around like variables and return them in functions and use them in other functions. When we pass a callback function as an argument to another function, we are only passing the function definition. We are not executing the function in the parameter. In other words, we aren’t passing the function with the trailing pair of executing parenthesis () like we do when we are executing a function.

And since the containing function has the callback function in its parameter as a function definition, it can execute the callback anytime.

**In case they are defined as closures in the other function its execution depends on the invocation in the run time.** If it Immediately Invocable Function then it will be executed immediately.

Note that the callback function is not executed immediately. It is “called back” (hence the name) First of all, function declarations are parsed and evaluated before any other expressions are

Use Named OR Anonymous Functions as Callbacks

It is somewhat like **variable closure whose function definition is set as per the programming need while closures are fixed defined**.

**12. Promises:**

<https://docs.angularjs.org/api/ng/service/$q>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

The Promise object is used for asynchronous computations.

A Promise represents a value which may be available now, or in the future, or never.

A pending promise can either be fulfilled with a value, or rejected with a reason (error). When either of these options happen, the associated **handlers** queued up by a promise's **then method** are called.

As the [Promise.prototype.then()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then) and [Promise.prototype.catch()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/catch) methods return promises, they can be chained.

**In promise we will normally see the callback function is being passed but since we don’t have the function definition to which it is being passed we can not predict when the callback will be executed.**

**Here you can assume that the parameters other than callback function may be used for the function’s core functionality which can be termed as asynchronous upon completion of which the callback will be executed.**

<https://www.promisejs.org/>

The core idea behind promises is that a promise represents the result of an asynchronous operation.

**Promise Object:**

new Promise( /\* **executor** \*/ function(**resolve**, **reject**) { ... } );

A function that is passed with the arguments resolve and reject. The executor function is executed immediately by the Promise implementation, passing resolve and reject functions (the executor is called before the Promise constructor even returns the created object). The resolve and reject functions, when called, resolve or reject the promise, respectively.The executor normally initiates some asynchronous work, and then, once that completes, either calls the resolve function to resolve the promise or else rejects it if an error occurred.

**To understand promise.then() method:**

<https://promisesaplus.com/>

“then” must return a promise

promise2 = promise1.then(onFulfilled, onRejected);

Chai as Promised's assertions return promise which is regular Chai assertion objects.

**Then method syntax:**

A promise must provide a then method to access its current or eventual value or reason.

A promise’s **then** method accepts two arguments:

**promise**.then(onFulfilled, onRejected)

doSomethingAsync().then(

function (result) {

result.should.equal("foo");

done();

},

function (err) {

done(err);

}

);

**Here doSomethingAsync() represents promise.**

**onFulFilled & onRejected are two callback functions denoted by** function (result)

function (err) respectively.

Alternative way with chai-as-promised:

**return** doSomethingAsync().should.eventually.equal("foo");

**Protractor APIs:**

<http://www.protractortest.org>

<http://www.protractortest.org/#/toc>

<https://github.com/cucumber/cucumber-js#promises>

<https://github.com/angular/protractor/blob/master/docs/control-flow.md>

**Control Flow:**

WebDriverJS (and thus, Protractor) APIs are entirely asynchronous. All functions return promises.

WebDriverJS maintains a queue of pending promises, called the control flow, to keep execution organized.

Rather than writing a long chain of promises, the **WebDriverJS library**

**promise manager** allows you to write code as if WebDriverJS had a synchronous, blocking API (like all of the other Selenium language bindings).

Using the promise manager comes at the cost of increased complexity.

Since the promise manager abstracts away the async nature of the API, it hides that you need to explicitly use a callback to break after a command executes.

Jasmine expectations are also adapted to understand promises. That's why this line works - the code actually adds an expectation task to the control flow, which will run after the other tasks:

expect(name.getText()).toEqual('Jane Doe');

However, the expect function in Mocha/Cucumber is not adapted to understand promises - that's why you'll need to use an assertion framework such as Chai as Promised. See [Choosing a Framework](https://github.com/angular/protractor/blob/master/docs/frameworks.md) for more information.

**WebElements**:

//To locate first matching web element

**ElementFinder element(locator);**

//To locate array of web elements matching the

**ElementArrayFinder element.all(locator);**

//To get the particular element from ElementArrayFinder

**get(index)**

**first()**

**last()**

//To get the number of elements in ElementArrayFinder

**count()**

//To return a new ElementArrayFinder with all elements that pass the filter function. The filter function receives the ElementFinder as the first argument and the index as a second arg.

element.all(locator).filter(filterFn(element, index))

**$$(cssSelector)**

Shorthand function for finding arrays of elements by css. element.all(by.css('.abc')) is equivalent to $$('.abc')

When you find elements in Protractor all actions are asynchronous.

When using CSS Selectors as a locator, you can use the shortcut $() notation:

$('my-css');

// Is the same as:

element(by.css('my-css'));

element.all() has several helper functions:

When using CSS Selectors as a locator, you can use the shortcut $$() notation:

$$('.selector');

// Is the same as:

element.all(by.css('.selector'));

**Locators**:

1. Extension to the webdriver.By:

id

name

className

css

xpath

linktext

partialLinkText

tagName

Js

2. ProtractorBy:

binding

exactBinding

Model

repeater

exactRepeater

cssContainingText

buttonText

partialButtonText

options

**Wait**

Represents a library of canned expected conditions that are useful for protractor, especially when dealing with non-angular apps.

Each condition returns a function that evaluates to a promise.

browser.wait(protractor.**ExpectedConditions**.elementToBeClickable(**this**.**getQuotebutton**), 5000);

You can define your own expected condition, which is a function that

takes no parameter and evaluates to a promise of a boolean.

var urlChanged = function() {

return browser.getCurrentUrl().then(function(url) {

return url === 'http://www.angularjs.org';

});

};

You may mix multiple conditions using and, or, and/or not. You may also mix these conditions with any other conditions that you write.

protractor.**ExpectedConditions.and(urlChanged,** protractor.**ExpectedConditions.textToBePresentInElement($('abc'),**

**'bar'),** protractor.**ExpectedConditions.elementToBeClickable(button));**

**protractor.ExpectedConditions**

elementToBeClickable

elementToBeSelected

textToBePresentInElement

textToBePresentInElementValue

visibilityOf

invisibilityOf

urlContains

titleContains

presenceOf

stalenessOf

alertIsPresent

**Assertions:**

<http://chaijs.com/api/>

Possible combinations of matcher functions:

**to.equal/eql/include/contain;**

**to.be.ok/true/false/null/undefined/NaN/a/an(instanceof)/empty/within/closeTo**

**to.exist**

to.have.any/all(keys/deep(property))/length(above/below/at(most))/string

<https://github.com/domenic/chai-as-promised>

<https://www.npmjs.com/package/chai-as-promised>

**Cucumber**:

<https://github.com/cucumber/cucumber-js/blob/master/docs/cli.md>

Note to Windows users: Use cucumber-js or cucumberjs instead of cucumber.js

In order to store and reuse commonly used CLI options, you can add a cucumber.js file to your project root directory.

The file should export an object where the key is the profile name and the value is a string of CLI options. The profile can be applied with -p <NAME> or --profile <NAME>. This will prepend the profile's CLI options to the ones provided by the command line. Multiple profiles can be specified at a time. If no profile is specified and a profile named default exists, it will be applied.s

**~/workspace/MyPro/node\_modules/cucumber/bin$ node cucumber.js ../../../features --tags @demo**

**Cucumber feature files in Intellij:**

Install following plugins in Settings> plugins:

Gherkin

Cucumber for java

Add following file type Registered pattern in Settings>Editor>File Types>Cucumber Scenario:

\*.feature

Cucumber Tags:

<https://docs.cucumber.io/tag-expressions/>

**Page Object Pattern:**

<https://semaphoreci.com/community/tutorials/using-page-objects-with-protractor-and-cucumber-in-angular-applications>

**Logging framework log4j:**

<http://www.tutorialspoint.com/log4j/>

**World:**

World is an isolated context for each scenario, exposed to the hooks and steps as **this**

This is what's known as a **destructuring** assignment, and it's a new feature of JavaScript 1.7 (and **ECMAScript 6**)

var ActionButton = require("sdk/ui/button/action").ActionButton; **is now**

var { ActionButton } = require("sdk/ui/button/action");

It seems silly in this example, as there's only one item being assigned. However, you'd be able to use this pattern to assign multiple variables at once:

{x, y} = foo;

Is the equivalent to:

x = foo.x;

y = foo.y;

This can also be used for arrays. For example, you could easily swap two values without using a temporary variable:

var a = 1;var b = 3;

[a, b] = [b, a];

<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

**Class in ECMAScript 6:**

<https://www.sitepoint.com/understanding-ecmascript-6-class-inheritance/>

class AnimalES6 {

constructor(name) {

this.name = name;

}

doSomething() {

console.log("I'm a " + this.name);

}

}

var lionES6 = new AnimalES6("Lion");

lionES6.doSomething();

**let in ECMAScript 6:**

The difference is scoping. var is scoped to the nearest function block and let is scoped to the nearest enclosing block (both are global if outside any block), which can be smaller than a function block.

Also, variables declared with let are not accessible before they are declared in their enclosing block. As seen in the demo, this will throw a ReferenceError exception.

Demo:

Show code snippet

Global:

They are very similar when used like this outside a function block.

let me = 'go'; // globally scoped

var i = 'able'; // globally scoped

However, global variables defined with let will not be added as properties on the global window object like those defined with var.

console.log(window.me); // undefined

console.log(window.i); // 'able'

Function:

They are identical when used like this in a function block.

function ingWithinEstablishedParameters() {

let terOfRecommendation = 'awesome worker!'; //function block scoped

var sityCheerleading = 'go!'; //function block scoped

}

**Block:**

Here is the difference. let is only visible in the for() loop and var is visible to the whole function.

function allyIlliterate() {

//tuce is \*not\* visible out here

for( let tuce = 0; tuce < 5; tuce++ ) {

//tuce is only visible in here (and in the for() parentheses)

//and there is a separate tuce variable for each iteration of the loop

}

//tuce is \*not\* visible out here

}

function byE40() {

//nish \*is\* visible out here

for( var nish = 0; nish < 5; nish++ ) {

//nish is visible to the whole function

}

//nish \*is\* visible out here

}

**Redeclaration:**

Assuming strict mode, var will let you re-declare the same variable in the same scope. On the other hand, let will not:

'use strict';

let me = 'foo';

let me = 'bar'; // SyntaxError: Identifier 'me' has already been declared

'use strict';

var me = 'foo';

var me = 'bar'; // No problem, `me` is replaced.

let can also be used to avoid problems with closures. It binds fresh value rather than keeping an old reference as shown in examples below.

DEMO

for(var i = 1; i < 6; i++) {

document.getElementById('my-element' + i)

.addEventListener('click', function() { alert(i) })

}

Code above demonstrates a classic JavaScript closure problem. Reference to the i object is being stored in the click handler closure, rather than the actual value of i.

Every single click handler will refer to the same object because there’s only one counter object which holds 6 so you get six on each click.

General workaround is to wrap this in an anonymous function and pass i as argument. Such issues can also be avoided now by using let instead var as shown in code below.

DEMO (Tested in Chrome and Firefox 50)

'use strict';

for(let i = 1; i < 6; i++) {

document.getElementById('my-element' + i)

.addEventListener('click', function() { alert(i) })

}

**Class:**

**'use strict'**;

**class** SrpPage {

constructor() {

**this**.**package** = *element*.*all*(By.repeater(**'ng-repeat="packageItem in $ctrl.packages | limitTo:itemsNgRepeatLimit.value track by packageItem.id"'**));

}

totalNumberOfPackages() {

**return this**.**package**.count();

}

}

***module***.exports = SrpPage;

**Logger:**

**let log4js** = ***require***(**'log4js'**);

**let** logger = **log4js**.*getLogger*();

Inside the class:

logger.trace(**'Value is: '**+text)

Some good to read:

<http://engineering.wingify.com/posts/e2e-testing-with-webdriverjs-jasmine/>

Soap UI:

Apache CXF stub:

(It creates Java implementation of the wsdl file)

Tools > Apache CXF

Package Name: (It should be appropriate to avoid [failed to localize]

error) main.java.ws.invoice.v1

Corresponding class files are loaded in the specified folder:

----------------------------------------------------------------------------------------------------------------------------------------------------------------->

**Maven:**

**Lifecycle consists of variety of phases**

**internalqa@truenorthlogic.com**

**----------------------------------------------------------------------------------------------------------------------------------------------------------------->**

**Creation & initialisation of the array of objects in java:**

A[] a = new A[] { new A("args"), new A("other args"), .. };

**Jason deserialisation meaning : Converting JSON to Java Object.**

**Getter classes:**

**Returns Users & Data**

**Users.java -> Users users <- readUsersData() <- InputStream(TestData.json)**

**Data.java -> Data data <- readTestData () <- InputStream(LoginData.json)**

@JsonDeserialize(as = EvaluationProgram.**class**)

**Will deserialise given json in the format as described in the given class.**

**Enum class TestData**

**Go Language:**

**Installable from:**

<https://golang.org/dl/>

After installation it will contain gofmt.exe that would be used by eclipse preferences to set the variables.

**Set up in eclipse:**

**Get following package executables:**

* The [gocode](https://github.com/nsf/gocode) tool. It is recommended to use the latest gocode version.
* The [guru](https://godoc.org/golang.org/x/tools/cmd/guru) tool.
* The [godef](https://github.com/rogpeppe/godef) tool.

go get github.com/nsf/gocode

go get golang.org/x/tools/cmd/guru

go get github.com/rogpeppe/godef